Iterables in ES6:

In JavaScript, an iterable is an object that can be iterated (looped) over, such as an array or a string. ES6 (ECMAScript 6) introduces a new way to iterate over iterable objects using the for...of loop and the Symbol.iterator method.

The for...of loop is used to iterate over the elements of an iterable object, such as an array or a string. Here is an example:

```
let colorArray = ["red", "green", "blue"];
for (let color of colorArray) {
  console.log(color);
}
// Output: red green blue
```

In the above example, the for...of loop iterates over the elements of the colorArray array and assigns the value of each element to the color variable in each iteration.

The Symbol.iterator method is used to create a custom iterator for an object. It is a method that returns an object that implements the next() method, which is used to iterate through the elements of the object. Here is an example:

```
let numbers = {
  *[Symbol.iterator]() {
    for (let i = 1; i <= 3; i++) {
      yield i;
    }
  }
};

for (let num of numbers) {
    console.log(num);
}
// Output: 1 2 3
```

In this example, the object numbers is defined with a generator function, which is a special type of function that uses the yield keyword to return a sequence of values. The Symbol.iterator method is used to create an iterator for the object, which can be used in a for...of loop to iterate through the values returned by the generator function.

ES6 also introduced the spread operator ... that allows to spread iterable objects into arrays or other iterable objects.

```
let colors1 = ["red", "green"];
let colors2 = ["blue", "yellow"];
let allColors = [...colors1, ...colors2];
console.log(allColors); // Output: ["red", "green", "blue", "yellow"]
```

Iterables and the related features, such as the for...of loop and the Symbol.iterator method, are powerful features that make it easier to work with arrays and other iterable objects in JavaScript, and provide a more expressive and readable way to iterate.

forEach in ES6:

The forEach() method in JavaScript is a built-in method that can be used to iterate over the elements of an array and perform a specific action on each element. In ES6 (ECMAScript 6), the forEach() method is available on the Array prototype and it can be used to iterate over any array-like object.

Here is an example of using the forEach() method to iterate over an array and print the value of each element:

```
let colorArray = ["blue", "red", "green"];
colorArray.forEach(function(color) {
  console.log(color);
});
// Output: blue red green
```

In this example, the forEach() method is called on the colorArray array, and it takes a callback function as an argument. The callback function is called for each element in the array, and it receives the current element as an argument.
You can also use an arrow function as callback function

```
let colorArray = ["blue", "red", "green"];
colorArray.forEach(color => console.log(color));
// Output: blue red greene
```

The forEach() method also accepts an optional second argument that can be used to specify this.

Filter in ES6:

The filter() method in JavaScript is a built-in method that can be used to create a new array with all elements that pass a certain test, specified by a callback function. In ES6 (ECMAScript 6), the filter() method is available on the Array prototype and it can be used to filter any array-like object.

Here is an example of using the filter() method to create a new array with all elements that are greater than 5:

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let filteredNumbers = numbers.filter(function(number) {
  return number > 5;
});
console.log(filteredNumbers); // Output: [6, 7, 8, 9, 10]
```

In this example, the filter() method is called on the numbers array, and it takes a callback function as an argument. The callback function is called for each element in the array, and it receives the current element as an argument. The function should return a boolean value indicating whether the current element should be included in the new array.

You can also use an arrow function as a callback function.

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let filteredNumbers = numbers.filter(number => number > 5);
console.log(filteredNumbers); // Output: [6, 7, 8, 9, 10]
```

The filter() method does not modify the original array, it creates a new array that contains the elements that pass the test specified by the callback function.